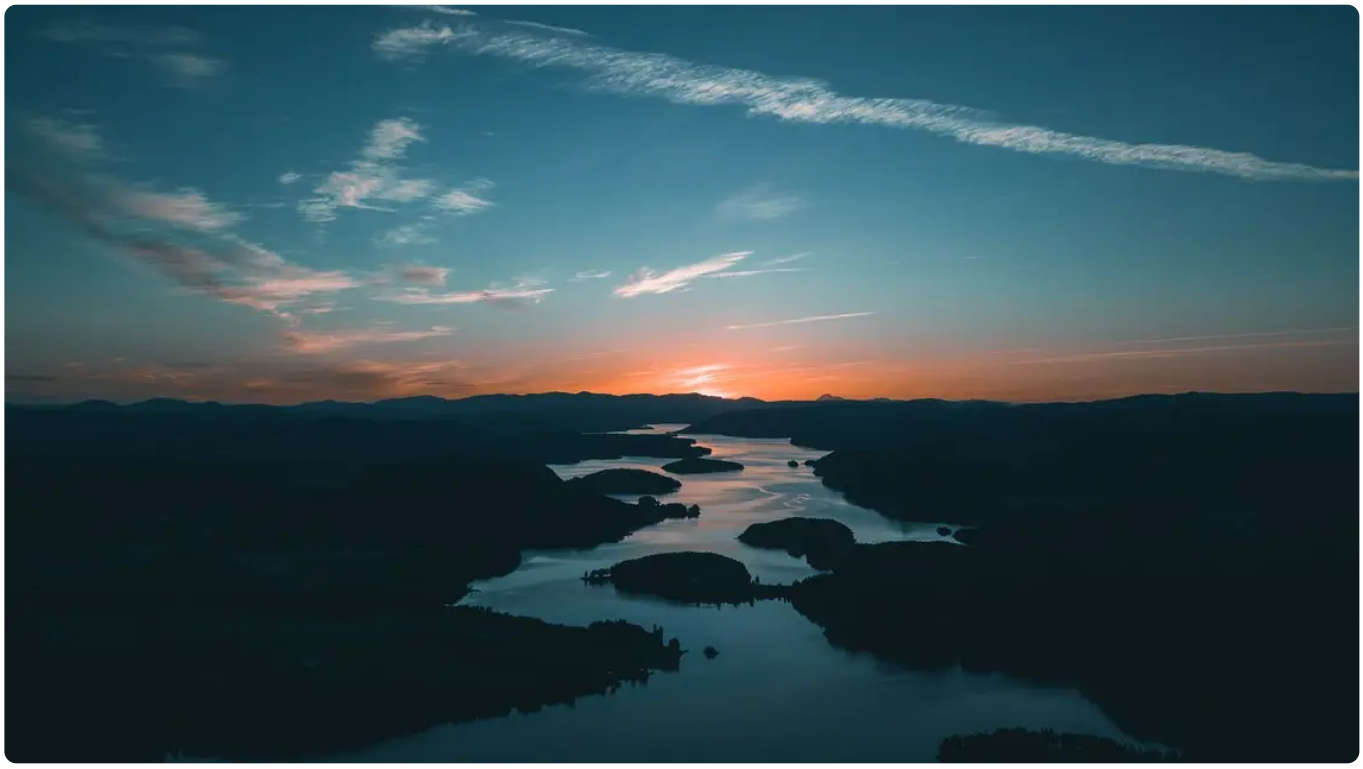
25 | 认证机制：应用程序如何进行访问认证？

2021-07-22 孔令飞

《Go 语言项目开发实战》 课程介绍 >



讲述：孔令飞
时长 17:08 大小 15.71M

你好，我是孔令飞，今天我们来聊聊如何进行访问认证。

保证应用的安全是软件开发的最基本要求，我们有多种途径来保障应用的安全，例如网络隔离、设置防火墙、设置 IP 黑白名单等。不过在我看来，这些更多是从运维角度来解决应用的安全问题。作为开发者，我们也可以从软件层面来保证应用的安全，这可以通过认证来实现。

这一讲，我以 HTTP 服务为例，来给你介绍下当前常见的四种认证方法：Basic、Digest、OAuth、Bearer。还有很多基于这四种方法的变种，这里就不再介绍了。

IAM 项目使用了 Basic、Bearer 两种认证方法。这一讲，我先来介绍下这四种认证方法，下一讲，我会给你介绍下 IAM 项目是如何设计和实现访问认证功能的。

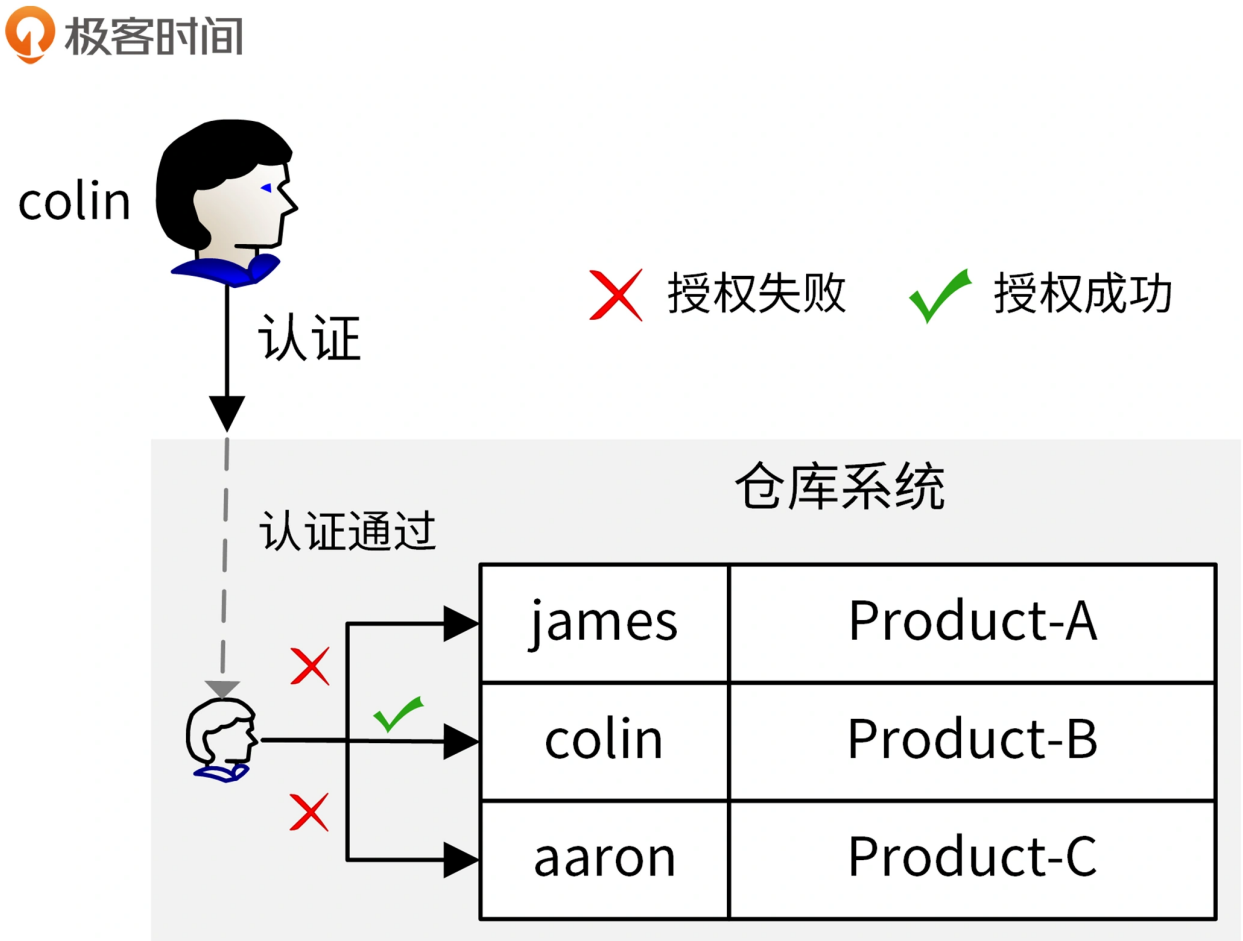
认证和授权有什么区别？

在介绍四种基本的认证方法之前，我想先带你区分下认证和授权，这是很多开发者都容易搞混的两个概念。

认证（Authentication，英文缩写 authn）：用来验证某个用户是否具有访问系统的权限。如果认证通过，该用户就可以访问系统，从而创建、修改、删除、查询平台支持的资源。

授权（Authorization，英文缩写 authz）：用来验证某个用户是否具有访问某个资源的权限，如果授权通过，该用户就能对资源做增删改查等操作。

这里，我通过下面的图片，来让你明白二者的区别：



图中，我们有一个仓库系统，用户 james、colin、aaron 分别创建了 Product-A、Product-B、Product-C。现在用户 colin 通过用户名和密码（认证）成功登陆到仓库系统中，但他尝试访问 Product-A、Product-C 失败，因为这两个产品不属于他（授权失

败），但他可以成功访问自己创建的资源 Product-B（授权成功）。由此可见：**认证证明了你是谁，授权决定了你能做什么。**

上面，我们介绍了认证和授权的区别。那么接下来，我们就回到这一讲的重心：应用程序如何进行访问认证。

四种基本的认证方式

常见的认证方式有四种，分别是 Basic、Digest、OAuth 和 Bearer。先来看下 Basic 认证。

Basic

Basic 认证（基础认证），是最简单的认证方式。它简单地将用户名:密码进行 base64 编码后，放到 HTTP Authorization Header 中。HTTP 请求到达后端服务后，后端服务会解析出 Authorization Header 中的 base64 字符串，解码获取用户名和密码，并将用户名和密码跟数据库中记录的值进行比较，如果匹配则认证通过。例如：

复制代码

```
$ basic=`echo -n 'admin:Admin@2021'|base64`
$ curl -XPOST -H"Authorization: Basic ${basic}" http://127.0.0.1:8080/login
```

通过 base64 编码，可以将密码以非明文的方式传输，增加一定的安全性。但是，base64 不是加密技术，入侵者仍然可以截获 base64 字符串，并反编码获取用户名和密码。另外，即使 Basic 认证中密码被加密，入侵者仍可通过加密后的用户名和密码进行重放攻击。

所以，Basic 认证虽然简单，但极不安全。使用 Basic 认证的唯一方式就是将它和 SSL 配合使用，来确保整个认证过程是安全的。

IAM 项目中，为了支持前端通过用户名和密码登录，仍然使用了 Basic 认证，但前后端使用 HTTPS 来通信，保证了认证的安全性。

这里需要注意，在设计系统时，要遵循一个通用的原则：**不要在请求参数中使用明文密码，也不要在任何存储中保存明文密码。**

Digest

Digest 认证（摘要认证），是另一种 HTTP 认证协议，它与基本认证兼容，但修复了基本认证的严重缺陷。Digest 具有如下特点：

绝不会用明文方式在网络上发送密码。

可以有效防止恶意用户进行重放攻击。

可以有选择地防止对报文内容的篡改。

摘要认证的过程见下图：

在上图中，完成摘要认证需要下面这四步：

1. 客户端请求服务端的资源。

2. 在客户端能够证明它知道密码从而确认其身份之前，服务端认证失败，返回401 Unauthorized，并返回WWW-Authentication头，里面包含认证需要的信息。
3. 客户端根据WWW-Authentication头中的信息，选择加密算法，并使用密码随机数 nonce，计算出密码摘要 response，并再次请求服务端。
4. 服务器将客户端提供的密码摘要与服务器内部计算出的摘要进行对比。如果匹配，就说明客户端知道密码，认证通过，并返回一些与授权会话相关的附加信息，放在 Authorization-Info 中。

WWW-Authentication头中包含的信息见下表：

极客时间

字段名	说明
username	用户名
password	密码
realm	服务器返回的realm，一般是域名
method	HTTP请求方法
nonce	服务器发给客户端的随机字符串
nc(nonceCount)	请求的次数，用于标记、计数，防止重放攻击
cnonce(clinetNonce)	客户端发送给服务器的随机字符串
qop	保护质量参数，一般是auth或auth-int，这会影响摘要的算法
uri	请求的uri
response	客户端根据算法算出的密码摘要值

虽然使用摘要可以避免密码以明文方式发送，一定程度上保护了密码的安全性，但是仅仅隐藏密码并不能保证请求是安全的。因为请求（包括密码摘要）仍然可以被截获，这样就可以重放给服务器，带来安全问题。

为了防止重放攻击，服务器向客户端发送了密码随机数 nonce，nonce 每次请求都会变化。客户端会根据 nonce 生成密码摘要，这种方式，可以使摘要随着随机数的变化而变化。服务端收到的密码摘要只对特定的随机数有效，而没有密码的话，攻击者就无法计算出正确的摘要，这样我们就可以防止重放攻击。

摘要认证可以保护密码，比基本认证安全很多。但摘要认证并不能保护内容，所以仍然要与 HTTPS 配合使用，来确保通信的安全。

OAuth

OAuth（开放授权）是一个开放的授权标准，允许用户让第三方应用访问该用户在某一 Web 服务上存储的私密资源（例如照片、视频、音频等），而无需将用户名和密码提供给第三方应用。OAuth 目前的版本是 2.0 版。

OAuth2.0 一共分为四种授权方式，分别为密码式、隐藏式、拼接式和授权码模式。接下来，我们就具体介绍下每一种授权方式。

第一种，密码式。密码式的授权方式，就是用户把用户名和密码直接告诉给第三方应用，然后第三方应用使用用户名和密码换取令牌。所以，使用此授权方式的前提是无法采用其他授权方式，并且用户高度信任某应用。

认证流程如下：

1. 网站 A 向用户发出获取用户名和密码的请求；
2. 用户同意后，网站 A 凭借用户名和密码向网站 B 换取令牌；
3. 网站 B 验证用户身份后，给出网站 A 令牌，网站 A 凭借令牌可以访问网站 B 对应权限的资源。

第二种，隐藏式。这种方式适用于前端应用。认证流程如下：

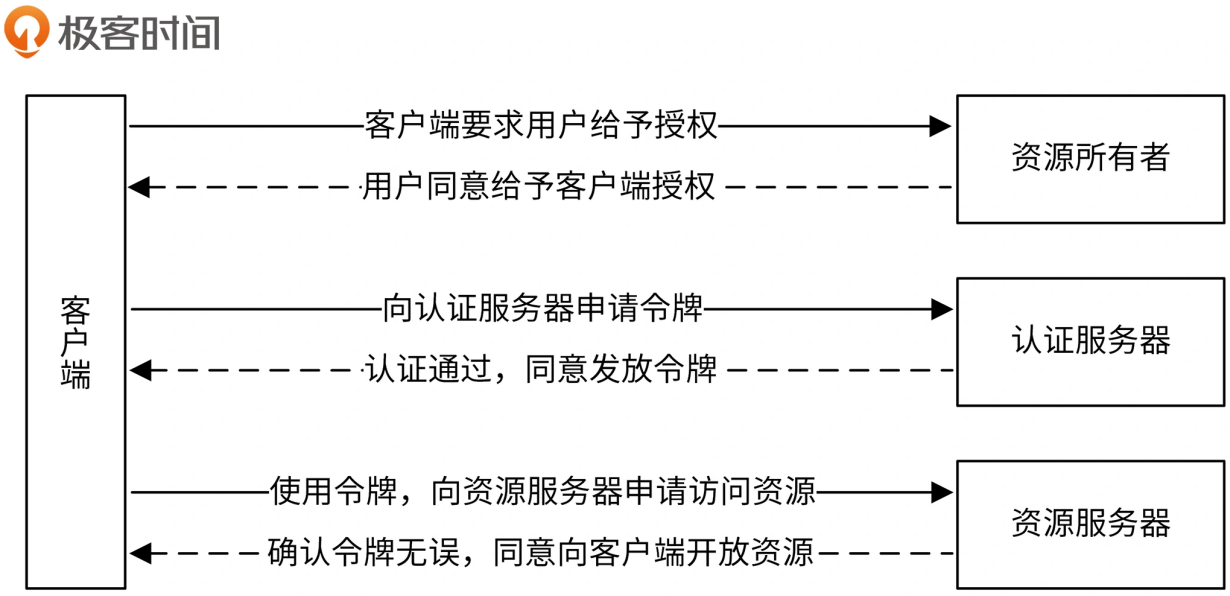
1. A 网站提供一个跳转到 B 网站的链接，用户点击后跳转至 B 网站，并向用户请求授权；
2. 用户登录 B 网站，同意授权后，跳转回 A 网站指定的重定向 redirect_url 地址，并携带 B 网站返回的令牌，用户在 B 网站的数据给 A 网站使用。

这个授权方式存在着“中间人攻击”的风险，因此只能用于一些安全性要求不高的场景，并且令牌的有效时间要非常短。

第三种，凭借式。这种方式是在命令行中请求授权，适用于没有前端的命令行应用。认证流程如下：

1. 应用 A 在命令行向应用 B 请求授权，此时应用 A 需要携带应用 B 提前颁发的 secretID 和 secretKey，其中 secretKey 出于安全性考虑，需在后端发送；
2. 应用 B 接收到 secretID 和 secretKey，并进行身份验证，验证通过后返回给应用 A 令牌。

第四种，授权码模式。这种方式就是第三方应用先提前申请一个授权码，然后再使用授权码来获取令牌。相对来说，这种方式安全性更高，前端传送授权码，后端存储令牌，与资源的通信都是在后端，可以避免令牌的泄露导致的安全问题。认证流程如下：



1. A 网站提供一个跳转到 B 网站的链接 +redirect_url，用户点击后跳转至 B 网站；
2. 用户携带向 B 网站提前申请的 client_id，向 B 网站发起身份验证请求；
3. 用户登录 B 网站，通过验证，授予 A 网站权限，此时网站跳转回 redirect_url，其中会有 B 网站通过验证后的授权码附在该 url 后；
4. 网站 A 携带授权码向网站 B 请求令牌，网站 B 验证授权码后，返回令牌即 access_token。

Bearer

Bearer 认证，也称为令牌认证，是一种 HTTP 身份验证方法。Bearer 认证的核心是 bearer token。bearer token 是一个加密字符串，通常由服务端根据密钥生成。客户端在请求服务端时，必须在请求头中包含`Authorization: Bearer <token>`。服务端收到

请求后，解析出 <token> ，并校验 <token> 的合法性，如果校验通过，则认证通过。跟基本认证一样，Bearer 认证需要配合 HTTPS 一起使用，来保证认证安全性。

当前最流行的 token 编码方式是 JSON Web Token（JWT，音同 jot，详见 JWT RFC 7519）。接下来，我通过讲解 JWT 认证来帮助你了解 Bearer 认证的原理。

基于 JWT 的 Token 认证机制实现

在典型业务场景中，为了区分用户和保证安全，必须对 API 请求进行鉴权，但是不能要求每一个请求都进行登录操作。合理做法是，在第一次登录之后产生一个有一定有效期的 token，并将它存储在浏览器的 Cookie 或 LocalStorage 之中。之后的请求都携带这个 token ，请求到达服务器端后，服务器端用这个 token 对请求进行认证。在第一次登录之后，服务器会将这个 token 用文件、数据库或缓存服务器等方法存下来，用于之后请求中的比对。

或者也可以采用更简单的方法：直接用密钥来签发 Token。这样，就可以省下额外的存储，也可以减少每一次请求时对数据库的查询压力。这种方法在业界已经有一种标准的实现方式，就是 JWT。

接下来，我就来具体介绍下 JWT。

JWT 简介

JWT 是 Bearer Token 的一个具体实现，由 JSON 数据格式组成，通过 HASH 散列算法生成一个字符串。该字符串可以用来进行授权和信息交换。

使用 JWT Token 进行认证有很多优点，比如说无需在服务端存储用户数据，可以减轻服务端压力；而且采用 JSON 数据格式，比较易读。除此之外，使用 JWT Token 还有跨语言、轻量级等优点。

JWT 认证流程

使用 JWT Token 进行认证的流程如下图：

具体可以分为四步：

1. 客户端使用用户名和密码请求登录。
2. 服务端收到请求后，会去验证用户名和密码。如果用户名和密码跟数据库记录不一致，则验证失败；如果一致则验证通过，服务端会签发一个 Token 返回给客户端。
3. 客户端收到请求后会将 Token 缓存起来，比如放在浏览器 Cookie 中或者 LocalStorage 中，之后每次请求都会携带该 Token。
4. 服务端收到请求后，会验证请求中的 Token，验证通过则进行业务逻辑处理，处理完后返回处理后的结果。

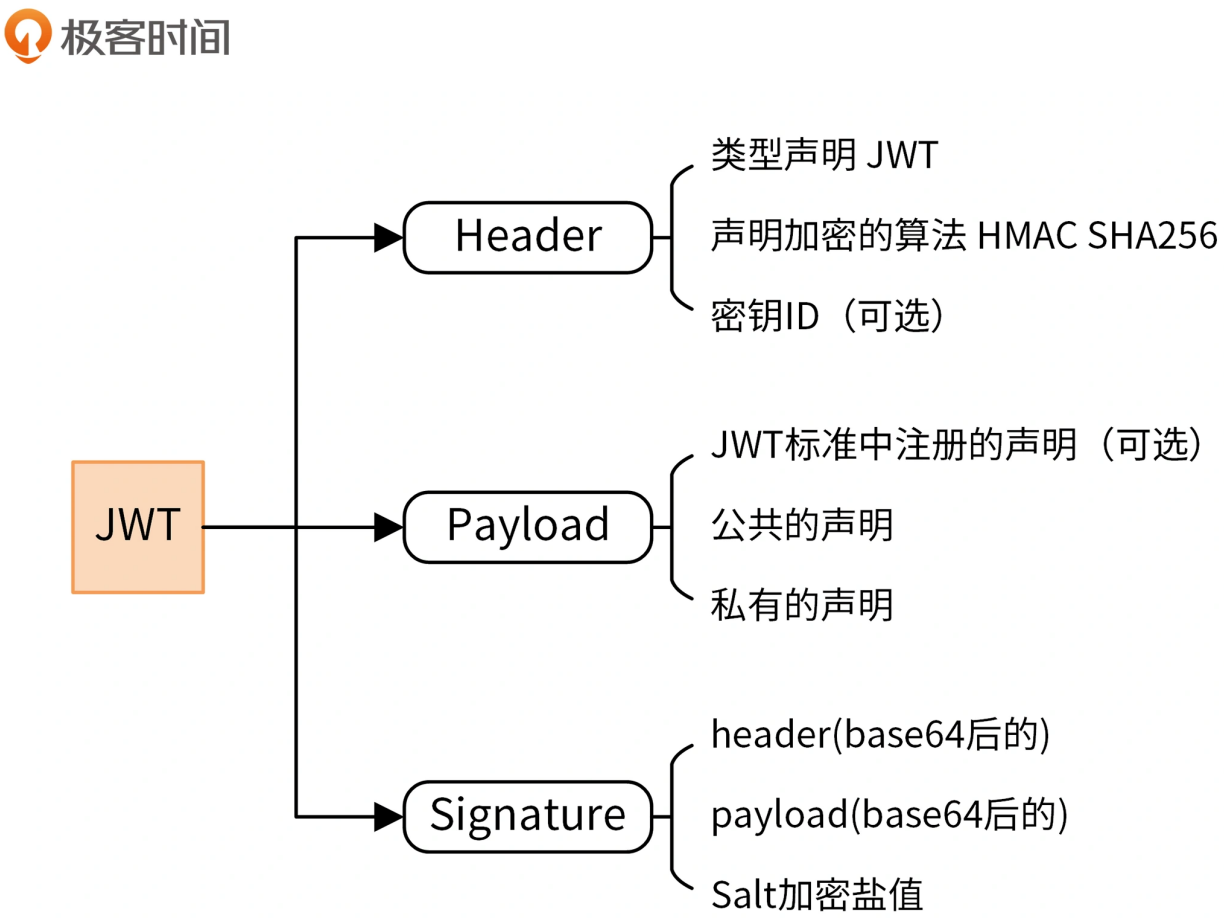
JWT 格式

JWT 由三部分组成，分别是 Header、Payload 和 Signature，它们之间用圆点 . 连接，并使用 Base64 编码，例如：

复制代码

```
1 JhdWQiOiJpYW0uYXV0aHoubWFybW90ZWR1LmNvbSIsImV4cCI6MTYwNDE1ODk4NywiaWF0IjoxNjA0
```

JWT 中，每部分包含的信息见下图：



下面我来具体介绍下这三部分，以及它们包含的信息。

1. Header

JWT Token 的 Header 中，包含两部分信息：一是 Token 的类型，二是 Token 所使用的加密算法。

例如：

复制代码

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

参数说明：

typ：说明 Token 类型是 JWT。

alg：说明 Token 的加密算法，这里是 HS256（alg 算法可以有多种）。

这里，我们将 Header 进行 base64 编码：

复制代码

```
$ echo -n '{"typ":"JWT","alg":"HS256"}'|base64
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

在某些场景下，可能还会有 kid 选项，用来标识一个密钥 ID，例如：

复制代码

```
{
    "alg": "HS256",
    "kid": "XhbY3aCrfjdYcP1OFJRu9xcno8JzSbUIvGE2",
    "typ": "JWT"
}
```

2. Payload（载荷）

Payload 中携带 Token 的具体内容由三部分组成：JWT 标准中注册的声明（可选）、公共的声明、私有的声明。下面来分别看下。

JWT 标准中注册的声明部分，有以下标准字段：

极客时间

字段名	描述
iss（Issuer）	JWT Token的签发者，其值应为大小写敏感的字符串或Uri
sub（Subject）	主题，sub可以用来鉴别一个用户
exp（Expiration Time）	JWT Token过期时间
aud（Audience）	接收 JWT Token的一方，其值应为大小写敏感的字符串或Uri，一般可以为特定的App、服务或模块。服务器端的安全策略在签发时和验证时，aud必须是一致的
iat（Issued At）	JWT Token签发时间
nbf（Not Before）	JWT Token生效时间
jti（JWT ID）	JWT Token ID，令牌的唯一标识符，通常用于一次性消费的Token

本例中的 payload 内容为：

复制代码

```
{
  "aud": "iam.authz.marmotedu.com",
  "exp": 1604158987,
  "iat": 1604151787,
  "iss": "iamctl",
  "nbf": 1604151787
}
```

这里，我们将 Payload 进行 base64 编码：

复制代码

```
$ echo -n '{"aud":"iam.authz.marmotedu.com","exp":1604158987,"iat":1604151787,
eyJhdWQiOiJpYW0uYXV0aHoubWFybW90ZWR1LmNvbSIsImV4cCI6MTYwNDE1ODk4NywiaWF0Ijox
NjA0MTUxNzg3LCJpc3MiOiJpYW1jdGwiLCJuYmYiOjE2MDQxNTE3ODd9
```

除此之外，还有公共的声明和私有的声明。公共的声明可以添加任何的需要的信息，一般添加用户的相关信息或其他业务需要的信息，注意不要添加敏感信息；私有声明是客户端和服务端所共同定义的声明，因为 base64 是对称解密的，所以一般不建议存放敏感信息。

3. Signature（签名）

Signature 是 Token 的签名部分，通过如下方式生成：将 Header 和 Payload 分别 base64 编码后，用 . 连接。然后再使用 Header 中声明的加密方式，利用 secretKey 对连接后的字符串进行加密，加密后的字符串即为最终的 Token。

secretKey 是密钥，保存在服务器中，一般通过配置文件来保存，例如：

```
# JWT 配置
jwt:
  realm: iam jwt # jwt 标识
  key: dfVpOK8LZeJLZHYmHdb1VdyRrACKpqoo # 服务端密钥
  timeout: 24h # token 过期时间(小时)
  max-refresh: 24h # token 更新时间(小时)
```

这里要注意，**密钥一定不能泄露。密钥泄露后，入侵者可以使用该密钥来签发 JWT Token，从而入侵系统。**

最后生成的 Token 如下：

复制代码

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJpYW0uYXV0aHoubWFybW90ZWR1LmNvb
```

签名后服务端会返回生成的 Token，客户端下次请求会携带该 Token。服务端收到 Token 后会解析出 header.payload，然后用相同的加密算法和密钥，对 header.payload 再进行

一次加密得到 Signature，并对比加密后的 Signature 和收到的 Signature 是否相同。如果相同则验证通过，不相同则返回`HTTP 401 Unauthorized`的错误。

最后，关于 JWT 的使用，我还有两点建议：

不要存放敏感信息在 Token 里；

Payload 中的 exp 值不要设置的太大，一般开发版本 2 小时，上线版本 7 天。当然，你也可以根据需要自行设置。

总结

在开发 Go 应用时，我们需要通过认证来保障应用的安全。认证，用来验证某个用户是否具有访问系统的权限，如果认证通过，该用户就可以访问系统，从而创建、修改、删除、查询平台支持的资源。业界目前有四种常用的认证方式：Basic、Digest、OAuth、Bearer。其中 Basic 和 Bearer 用得最多。

Basic 认证通过用户名和密码来进行认证，主要用在用户登录场景；Bearer 认证通过 Token 来进行认证，通常用在 API 调用场景。不管是 Basic 认证还是 Bearer 认证，都需要结合 HTTPS 来使用，来最大程度地保证请求的安全性。

Basic 认证简单易懂，但是 Bearer 认证有一定的复杂度，所以这一讲的后半部分通过 JWT Token，讲解了 Bearer Token 认证的原理。

JWT Token 是 Bearer 认证的一种比较好的实现，主要包含了 3 个部分：

Header：包含了 Token 的类型、Token 使用的加密算法。在某些场景下，你还可以添加 kid 字段，用来标识一个密钥 ID。

Payload：Payload 中携带 Token 的具体内容，由 JWT 标准中注册的声明、公共的声明和私有的声明三部分组成。

Signature：Signature 是 Token 的签名部分，程序通过验证 Signature 是否合法，来决定认证是否通过。

课后练习

1. 思考下：使用 JWT 作为登录凭证，如何解决 token 注销问题？

2. 思考下：Token 是存放在 LocalStorage 中好，还是存放在 Cookie 中好？

欢迎你在留言区与我交流讨论，我们下一讲见。

分享给需要的人，Ta订阅后你可得 24 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 24 | Web 服务：Web 服务核心功能有哪些，如何实现？

下一篇 26 | IAM项目是如何设计和实现访问认证功能的？

更多课程推荐

精选留言 (7)

写留言

8.13.3.27.30
2021-07-22

根据文中认证和授权的解释、我理解Oauth应该属于授权？而不是认证

作者回复: 其实都属于

1 2

pedro
2021-07-22

问题1：JWT中可新增一个 valid 字段用于表示 token 是否有效，注销后则无效。
问题2：token还是存储在cookie中比较好，可由服务端保存，localstorage在纯前端，中很容易泄露。

作者回复: 满分！

2

如来神掌
2021-07-22

方案挺多，如下:
1、最简单的，客户端直接删除token，但有弊端
2、token计入数据库，登录后对比，注销时删除
3、注销后记录到黑名单，下次登录后清楚黑名单
4、直接记到Redis，注销后删除，登录后存入，类似数据库...
展开

作者回复: 老哥回答的很赞。主要是删除token在客户端和服务端的缓存

1

宙斯
2021-07-30

digest认证中， “WWW-Authentication头中包含的信息见下表：” ，这里的下表，应该没有password吧，否则密码被泄露了
展开

作者回复: 感谢反馈，我们更新下

helloworld
2021-07-25

“Payload 中的 exp 值不要设置的太大，一般开发版本 2 小时，上线版本 7 天”，开发版本为什么比上线版本少这么多，基于什么考虑的呢

展开

作者回复: 这里写反了，我纠正下。线上版一般是2h，开发版随意。2h也需要你根据实际需要评估哈，如果你没任何需求可以先设置为2h

kitten
2021-07-25

用jwt就是为了节约服务端缓存。jwt token一旦颁发了，在有效期内无法注销。如果在服务端存了这个jwt的缓存，大可不必使用jwt，随机生成一个uuid给客户段就可以做登录凭证了。所以不知道老师出这个题目的目的是什么🤔？

作者回复: jwt可以节约服务端资源，但这不是用jwt的唯一目的。

服务端不会直接缓存token。

用UUID做凭证有很多问题：
1. 没有过期时间
2. UUID不包含用户信息
3. UUID可以伪造，Token有加密算法加密一些信息，更难伪造

lark
2021-07-22

在jwt的payload里加一个随机数字段，过期或者失效就重新生成一个随机数。

作者回复: 删除客户端的缓存就可以了